Google Summer Of Code

LabLua

Interrupt-based drivers and libraries for Ceu-Arduino

Milestone Report 2

Student:
Guilherme Simas

Mentor:
Francisco Santanna

June 27, 2017

Contents

1 Introduction

The goal of this report is to show the progress on the project from the past two weeks, comparing it with the expectation from the previous report in order to avaliate the progress made. In the first section the progress from previous report will be summarized. Following the summary, expectations for the proposed two-week period (June 23th) will be listed, followed by what was achieved.

This document concludes by discussing the next steps forward.

2 Previous Progress

On the previous, report, the modules chosen for development were presented along with a brief description of the motivation behind each one. The modules were **Analog I/O**, **SPI**, **Serial**, **External RTC** and **EEPROM**.

3 Milestone Objective

The goal for the current two-week period according to the planning was to deepen in the studies of the hardware and example applications in order to have a better understanding of what makes each module freeze and have all the necessary tools to begin proposing solutions. However, the team decided it was best to pivot this approach.

The new path will be to deal with each module individually. As such, the work from the next subsequent predicted periods (according to the planning) would be distriuted along the other periods, and 4 weeks would be gained to compensate for the redistributed work.

The goal for this period is now to study the **hardware**, the reason for the freezing behaviour in the **current Arduino library implementation** of the Analog I/O module, the **tools** provided from the development environment and, finally, to implement a **C version** of the API.

4 Progress

In order to develop the API, there was a need to study the implementation and how it behaves, so that, alongside the hardware information obtained from the datasheet, it would be possible to start developing a solution. The Analog I/O module provided in the Arduino API involves reading an analog value, which is done by a dedicated hardware, the **Analog To Digital Converter (ADC)** and writing an analog value, which is essentially outputting a PWM wave, since the hardware is only able to output a on or off digital voltage.

Writing an analog value is already implemented in the Ceu-Arduino current state as emitting a PWM output wave, ans therefore requires no reimplementation. The API needs only to reimplement reading the analog value by interfacing with the dedicated ADC hardware.

4.1 Hardware

The dedicate ADC hardware present in the microcontroller consists of a single dedicated hardware for the conversion which serves all analog input pins. It can read only one pin at a time since there is only one converter logic. The hardware counts with a multiplexer in order to select which pin is being read.

In order to perform a conversion, the software must first configure the ADMUX register, which control the voltage reference and the pin selection, and the **Analog to Digital Status and Control Register (ADSCRA)** responsible for turning the hardware on and enabling interrupts.

After the setup, setting the Analog to **Digital Start Conversion (ADSC)** bit in ADCSRA starts a conversion. When the conversion is done, the hardware will clear the register, and trigger an interruption if the **Analog to Digital Interrupt Enable (ADIE)** bit is set in ADCSRA.

After the conversion is done, the result will be available in the **ADC Data Registers Low (ADCL) and High (ADCH)**.

4.2 Current Implementation

The current library exports a function called analogRead() which takes in, as a parameter, the pin for which to read the value. The function sets the values in ADMUX (based on the pin and a default value for the reference) and sets ADSC. It then polls ADSC, freezing the application while ADSC is set. After it it cleared by the hardware, the function returns the result of the conversion stored in ADCH and ADCL.

4.3 Tools

AVR provides, along with a detailed datasheet on the microcontroller, a C compiler and libraries to better interface with the hardware. The most important things exported in the libraries, for the means of this API, are the register definitions and the interrupt service routines support. The library provides access to the registers memory addresses by name and also makes available functions for setting and clearing bits, also defined. Furthermore, it provides easy definition of **Interrupt Service Routines (ISR)** and attachment to the hardware's interrupt vectors. The user needs only to define a function with the name ISR and pass in as a parameter the interrupt vector number. The avr compiler will associate that routine with the corresponding interrupt.

4.4 C Version API

In order to reimplement the API using interrupts, the analogRead() function was divided unto 3 subfunctions: **begin()**, **available()** and **read()**. The new API also counts with a global state variable that keeps traack of the hardware state regarding a conversion, as well as an ISR attached to the ADC interrupt vector.

begin() sets up all the configuration registers alike the original implementation, with the only addition that it sets ADIE. It ends by setting ADSC in order to start the conversion. This function sets the value of the global state variable to reflect that a conversion is now in progress.

available() returns whether or not a conversion is still in progress, by checking the global state variable.

read() returns the conversion result in ADCH and ADCL, much like the original API.

Finally, the ISR, which is triggered when the conversion is done, sets the global state variable to reflect that a conversion is no longer in progress and, thereore, the result is available.

In order to perform a read on an analog pin using this API, the user must call begin() passing in the analog pin number as a parameter, then, after available() returns true, a call to read() returns the result wanted.

5 Conclusion and steps forward

The C version of the API proved to be a good learning tool and step towards the Ceu API. The goal to have a functioning version of the API, implemented in C, was achieved.

The next two-weeks period will be focused on the development of the Ceu version of the API.